Basic Comparison of High-Level Programming Languages

Computer Science / IT

Mentor: Jules Kouatchou
Code 606.0

Alexander Medema
Hope COLLEGE
HOLLAND, MICHIGAN

Abstract

Fourteen simple test cases were used to compare Python, Julia, Java, Scala, IDL, R, and Matlab, with Fortran and C included as a baseline. The test cases were implemented from the angle of a novice programmer who is not familiar with the optimization techniques available in the languages. The tests aimed to highlight the strengths and weaknesses of each language rather than to claim one language's superiority to the others. Measurements recorded the elapsed time to complete the same test case operation with each of the different languages. The performance was investigated in four main categories: loops and vectorization, string manipulations, numerical calculations, and input/output. Specific measurements tested the speed of memory access, recursion, file processing, matrix calculations, iterative solvers, and other common applications in scientific computing. No single language outperformed the others in all of the tests. Expected trends were demonstrated, such as which languages offer faster performance when using iteration versus vectorization, and that intrinsic functions operate more quickly than equivalent inline code. The performance of numerical calculations in each language was found to significantly depend on the specific task. Performance of I/O operations was found to depend more strongly on hardware resources than the language used. The results can serve as a reference for programmers who wish to determine which language(s) may be suitable for specific applications.

Introduction

Five of the tests are presented. Each was run on an Intel Xeon Haswell processor node of NASA's Discover supercomputer. Each node has 28 cores (2.6 GHz each) and 128 GB of available memory. The Python, Java, and Scala tests were also run on a Mac computer with an Intel i7-7700HQ (4 cores, 2.8GHz each) with 16 GB of available memory. Each measurement was taken in seconds to four digits of precision. Values below 0.0001 were considered to be 0.

I/O with NetCDF Files

- Data was read from a collection of 7305 NetCDF files.
- This type of operation is used by scientists to post-process files generated by models.
- Psudocode for the test reads:

```
Loop over the years
    Obtain the list of NetCDF files
    Loop over the files
        Read the variable (longitude/latitude/level)
        Compute the zonal mean average (new array of latitude/level)
        Extract the column array at latitude 86 degree South
        Append the column array to a "master" array (or matrix)
```

Language	Elapsed time (Discover)	Elapsed Time (Mac)
Python	660.8084	89.1922
Julia	787.4500	
IDL	711.2615	
R	1220.222	
Matlab	848.5086	

- The Mac has a solid-state drive, which likely allowed its test run more quickly than when using Discover's hard drives.

Look and Say Sequence

- The look and say sequence of order 48 was found, starting with "1223334444."
- This tested languages' handling of strings.

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		251.1905	126.0252
Java		0.1211	0.1543
Scala		0.2170	0.2040
IDL		1612.4277	
Matlab		exceeded time limit	
Fortran	gfortran -O3	0.0120	
C	gcc -Ofast	182.4500	

Copying Arrays

- Rows and columns of a 9000×9000×3 array were copied into new locations.
- Code that used loops (iterative) was compared with vectorized code.
- This test demonstrated each language's speed in accessing non-contiguous memory locations.

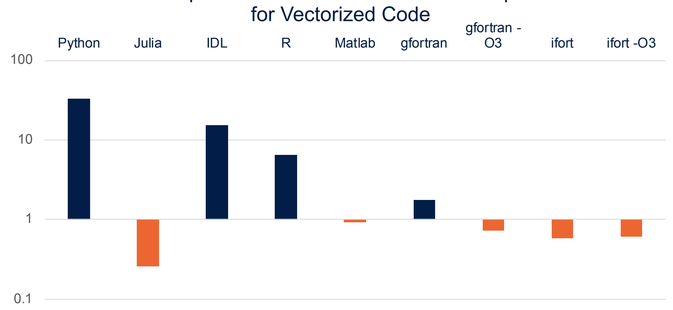
Loops

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		52.5485	60.2338
Julia		0.2359	
Java		0.5390	0.4190
Scala		0.7320	0.5150
IDL		19.4499	
R		74.3480	
Matlab		0.8461	
Fortran	gfortran -O3	0.2240	
C	gcc -Ofast	0.3100	

Vectorized

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		1.6078	1.8077
Julia		0.9191	
IDL		1.2643	
R		11.4400	
Matlab		0.9188	
Fortran	gfortran -O3	0.3120	

Ratio of Elapsed Time for Iterative Code to Elapsed Time for Vectorized Code



Fast Fourier Transform

- The Fast Fourier Transform (FFT) and absolute value of a 20000×20000 matrix of complex values were taken.
- Only intrinsic FFT functions were used.
- The FFT algorithm is commonly applied in science and engineering.

Language	Elapsed time (Discover)	Elapsed Time (Mac)
Python	34.7400	55.9375
Julia	20.751	
IDL	70.8142	
R	261.5460	
Matlab	10.66232	

Fibonacci Sequence

- The elapsed time when recursively calculating the 45th Fibonacci number was measured.

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		847.9716	800.0381
Julia		3.787	
Java		4.8192	5.0130
Scala		5.1400	5.7720
IDL		304.2198	
R		0.0100	
Matlab		149.9634	
Fortran	gfortran -O3	0	
C	gcc -Ofast	2.2000	

Laplace-Jacobi Solver

- A 4th-order finite difference scheme[2] was used to approximate a solution of the 2-D Laplace equation on a 200×200 grid.
- Two versions of the Jacobi iterative solver were implemented: looping over all grid points, and vectorization.
- Approximation of differential equation solutions is common in scientific computing.

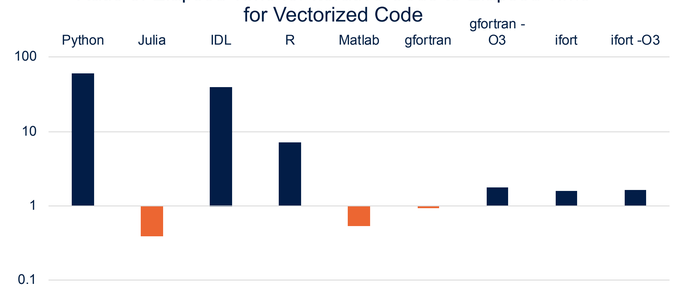
Loops

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		2437.8560	2666.3496
Julia		16.1651	
Java		5.2220	4.7530
Scala		5.7380	5.2830
IDL		1127.1094	
R		2414.1030	
Matlab		8.6276	
Fortran	gfortran -O3	8.8930	
C	gcc -Ofast	3.1900	

Vectorized

Language	Option	Elapsed Time (Discover)	Elapsed Time (Mac)
Python		1.6078	22.0945
Julia		0.9191	
IDL		1.2643	
R		11.4400	
Matlab		0.9188	
Fortran	gfortran -O3	0.3120	

Ratio of Elapsed Time for Iterative Code to Elapsed Time for Vectorized Code



Findings

- No single language outperformed the others in all tests.
- Python (and Numpy), IDL, and R consistently ran more quickly when vectorized compared to when using loops.
- With Julia, loops ran more quickly than vectorized code.
- Matlab did not appear to change significantly in performance when using loops versus vectorization in a case that involved no calculations. When calculations were performed, vectorized Matlab code was faster than iterative code.
- Matlab's intrinsic FFT function ran the most quickly.
- Java and Scala appeared to have notable performance relative to the other languages when manipulating large strings.
- R appeared to have notable performance relative to the other languages when using recursion.
- Languages' performance in numerical calculations relative to the others depended on the specific task.
- While some of the languages ran the I/O test more quickly than others, running the test on a local Mac instead of Discover resulted in the largest performance gain. Discover uses hard drives, whereas the Mac has a solid-state disk. This indicated that hardware had a larger impact on I/O performance than the language used.

Acknowledgements

This work funded by Michigan Space Grant Consortium, NASA grant #NNX15AJ20H.

References, more tests and data, and upcoming source code are on modelingguru.nasa.gov: